Exploring Embedded Path Capacity Estimation in TCP Receiver

Cesar Marcondes[1], M. Y. Sanadidi, Mario Gerla
Computer Science Department
University of California (UCLA)
Los Angeles, USA
{cesar,medy,gerla}@cs.ucla.edu

Magnos Martinello[2] and Ramon S. Schwartz
Computer Science Department
Universidade Federal do Espırito Santo (UFES)
Vitoria, Brazil
{magnos,ramon}@inf.ufes.br

***Abstract*— Accurate estimation of network characteristics, such as capacity, based on non-intrusive measurements is a fundamental desire of several applications. For instance, P2P applications that build overlay networks can use path capacity for optimizing network performance. We present a simple technique to estimate end-to-end Internet paths capacity simply by making adequate inferences at the TCP receiver behavior. Our proposed method does not require access or permission at remote machines. An off-the-shelf Linux kernel is used to implement the method and provide precise measurements. In addition, a large number of experiments in a high speed environment are used to i) validate, ii) show the accuracy and iii) evaluate path capacity heterogeneity of some Internet paths. Applications of our method include improving TCP throughput using the receiver advertised window, and identifying the location of the narrow link (i.e. bottleneck) on an Internet path.**

Keywords- Measurements, Monitoring, Network Performance Evaluation, Inline TCP Estimation.

I. INTRODUCTION

Path characteristics estimation is a fundamental desire of several applications. For instance, streaming applications using a peer-to-peer backbone (such as Skype, P2PLive) have certain requirements of end-to-end path bandwidth and delay. Thus, in order to help applications achieve their requirements, it is useful to estimate capacity, delay, jitter and loss patterns of Internet paths. Such path characteristics estimations can be gathered in various ways, using either active, passive or mixed network measurements.

In the case of active measurements, it is up to the application to send special probes from source(s) to destination(s), hence incurring the use of extra probing bandwidth. On the other hand, network measurements can be performed in a passive [11] mode, which relies on traces to infer characteristics of a path. Finally, a third mixed technique can be achieved by inferring characteristics through inspection of the application own traffic pattern, without inserting any extra probes on the path.

The purpose of our work is to present a simple technique to embed end-to-end path capacity estimation into applications, thus using the mixed methodology (i.e. without sending probes while receiving useful data). The core idea relies on the observation that path capacity estimations can be obtained simply by making adequate inferences on the TCP traditional behavior, more precisely, by measuring packets dispersion during a TCP slow start phase. If the packets were sent back-to-back (packet pairs), such dispersion may represent the capacity of the narrow link on a path. The proposed technique in this paper focuses on the receiver side of a TCP connection, allowing measurements to be executed in the Internet with no requirement of having access and/or permission to remote machines (i.e. Internet file servers). Moreover, since the measurement of packets dispersion can produce either over-estimation or under-estimation of path capacity due to cross-traffic at some link [7], our technique requires that a certain packet pair interarrival have the minimal round-trip delay among all packet pairs. A possible scenario in which our method turns beneficial is the case of an open source file server that wishes to identify path capacity to a number of client systems. In this case, the server is a TCP receiver, and the measurement function will be carried out at such a receiver. A server deploying our measurement method may improve TCP throughput by setting its TCP "advertised window" according to the estimated path capacity. Consequently, it can limit the amount of traffic in flight and relieve buffer congestion at the bottleneck router. Another possible beneficial deployment, in combination with traceroute and monarch [6], is for identifying the bottleneck link location on a path.

The remainder of this paper is structured as follows. In section 2, our attention is first devoted to related techniques for capacity estimation, then we focus on the proposed TCP capacity inference methods. Section 3 describes the Linux Kernel instrumentation in detail. In Section 4, we report experimental results to i) validate and compare the performance of the TCP inference methods, ii) show the accuracy of the Internet capacity estimation and iii) explore the heterogeneity of the Internet paths. Section 5 presents two applications making use of the embedded TCP receiver feature and Section 6 concludes the paper.

II. CAPACITY ESTIMATIONS TECHNIQUES

One of the earliest methods to estimate path capacity was described in [1]. The method called bprobe, relies on the idea that if two packets are traveling together, they are to be queued as a pair at the bottleneck link, then the inter-packet spacing will be proportional to the service time of the bottleneck. This work presented an early version of the packet pair techniques (Figure 1).

[1] Cesar Marcondes is supported by CAPES/Brazil #1283-02-2
[2] M. Martinello is supported by CNPq/Brazil Pos-Doc Junior (PDJ)

Later, in [8] the authors suggested a robust capacity estimation technique called PBM (Packet Bunch Sizes). PBM technique works by stepping through an increasing series of packet bunch sizes [1]. For each sample, the bottleneck estimation is computed based on the receiver trace. After the bottleneck distribution is constructed, the final estimated value is obtained by determining the maximum value in the density function. If two modes are similar and sufficiently separated, it suggests a change in the service rate of the bottleneck.

The meaning of the multiple modes in packet pair dispersions and packet trains was elaborated further in [3], where it was shown that the strongest mode in the multimodal dispersion distribution may not correspond to the path capacity, but to an under or over-estimated capacity. A capacity estimation methodology (Pathrate) was developed based on this analysis. Pathrate requires a number of packet pairs (with packets of variable size) to identify a set of potential "capacity modes". Then, in a second phase, "packet trains" are sent to estimate the so called "Asymptotic Dispersion Rate" or ADR. ADR corresponds to a measure of the average statistical multiplexing of the path. The ADR provides a hint about which capacity local modes to reject. Finally, Pathrate chooses the mode that has the strongest and narrowest mode from the non-rejected ones.

There are other techniques not based on dispersion of packet pairs. In [4], a pathchar tool is proposed using the variation of the round-trip delay as the packet size increases. This technique, based on the generation of ICMP replies from routers, is known to have scalability and accuracy problems. In fact, it tries to estimate the capacity of every link on the path in order to estimate the end-to-end path's capacity, resulting in high overhead to accomplish the capacity estimation.

Although these techniques are well-know and widely used, we can observe several limitations. First, some of them require the execution of measurement code at both ends of the measured path [3]. This constraint limits the applicability of these tools in just a few paths where the user has access at both the sender and the receiver. Second, some of the tools depend on ICMP probing packets. Such traffic is often blocked or handled in different processing path than TCP traffic. Third, the time to converge to an accurate metric is also a drawback since several of the methods require a thorough statistical analysis in order to ensure a reliable capacity estimate, taking in worst cases several minutes. And at last, all of the capacity estimation techniques are based on active measurements, requiring special probing flows to be inserted into the network.

CapProbe [7] tackles the convergence speed problem by filtering out packet pairs according to a simple rule: "packet pairs with minimal end-to-end delays are sufficient to estimate consistently a narrow link capacity". This way, it rules out packet pairs impacted by cross-traffic, and no posterior statistical analysis is necessary to obtain an accurate estimate, providing a faster estimation time relative to previous methods.

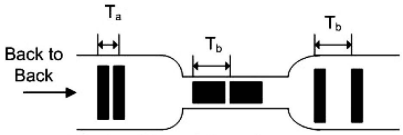


Figure 1 – Packet Pair Dispersion and its Relationship to the Narrow Link Capacity

The main idea underlying CapProbe is that at least one of the two probing packets must have queued if the dispersion at the destination has been distorted from that corresponding to the narrow link capacity. This means that for samples that estimate an incorrect value of capacity, the sum of the delays of the packet pair packets, which CapProbe [7] called "the delay sum", includes cross-traffic induced queuing delay. This delay sum will be larger than the "minimum delay sum", which is the delay sum of a sample in which none of the packets suffer cross-traffic induced queuing. The dispersion of such a packet pair sample is not distorted by cross-traffic and will reflect the correct capacity. Based on this observation, CapProbe calculates delay sums of all packet pair samples and uses the dispersion of the sample with the minimum delay sum to estimate the narrow link capacity. Thus:

$$Capacity(C) = \frac{Packet\_Size(L)}{NonQueued\_PPair\_Dispersion(Tb)}$$

In summary, CapProbe is the main candidate to be embedded in applications. Thus, the main question that arises and motivation of this work is: **"Could we embed such a path capacity estimation within TCP receiver, with minimal changes to TCP?"**. This approach would be general enough to cover a wide range of applications (i.e., estimate path capacity while an application is downloading a page on the web), it would require no additional probing traffic. In addition, it would also overcome potential impediments of previous methods. For example, non-cooperation by one end of a connection or ICMP inaccuracy by using TCP, while it inherits the speeds up convergence to a capacity estimate as discussed in CapProbe summary.

A. TCP Inference Methods for Capacity Estimation

The TCP inference methods we present in this work rely on identifying pairs of data packets sent "back to back" from the sender.

[1] Packet bunch or packet train mean back-to-back packets with size greater or equal to two.

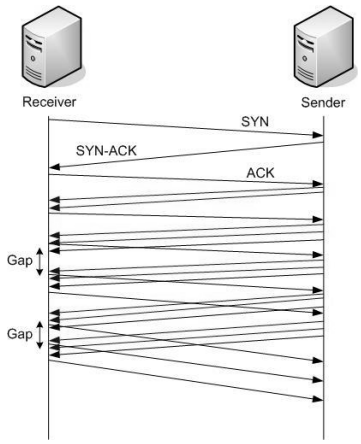


Fig. 2(a) Short Regulated Rate (SRR)

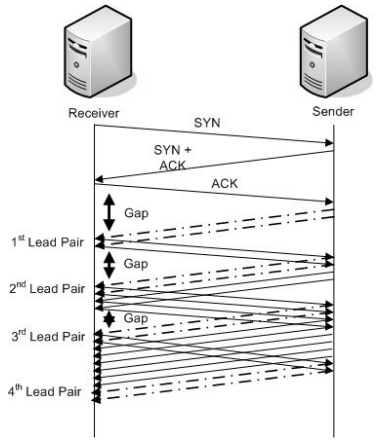


Fig. 2(b) Lead Packet Pair (LPP)

In order to perform such identification, the methods have to be aware of the Slow Start (SS) phase, in which, every Acknowledgment from the TCP receiver triggers the transmission of a packet pair "back to back" from the sender. Although SS generates packet pairs in abundance, it also creates packet train patterns at each RTT cycle (that is a sequence of more than 2 packets, that are sent back to back). Assuming no cross traffic, the train packets, except its first and second packets, will queue, impeding the CapProbe algorithm to find the minimum delay sum, a behavior that is referred to as self-interference. We handle the trains obstacle by two methods (1) Short Regulated Rate (SRR), and (2) Lead Packet Pair (LPP) in each RTT cycle.

The Short Regulated Rate (SRR) method induces the sender to transmit back-to-back packets, or packet pairs more often. The basic idea is to use a generalized Delayed Ack scheme in SS, beginning with the second cycle of a connection. It is important to note that normally Delayed Acks are not used in Slow Start. The purpose of using Delayed Acks is to force time gaps in the sender transmission sequence, and thereby allowing the receiver to capture a relatively larger number of packet pairs. A parameter of this scheme is the number of packets to accumulate before sending the Delayed Ack. In our experiments, the parameter value was set to 2, despite; it can be generalized to higher values if needed. Higher values are needed when the receiver is not getting enough packet pairs. On the other hand, a larger value would slow down convergence to an estimate since it will delay much more the generation of the next set of packets.

Figure 2(a) describes a sequence of packets and Acks illustrating the Short Regulated Rate (SRR) method, which is called 'short' because it is restricted to the first 20 packet pairs identified at the receiver. The short period of time in which this algorithm is in effect is based on the fact that 20 samples has been in most cases sufficient for Capprobe [7] to converge to an accurate estimate. Doing a simple worst-case calculation, assuming an RTT of 50 ms, the time taken to collect 20 samples would be of only 1 second. Subsequently to this short phase, there is no more "ACKs regulation" and TCP will resume its normal Slow-Start phase without delayed Acks

The second method attempts to identify the sender RTT cycle boundaries, and extract the leading packet pair at the beginning of each cycle. The leading packet pair is less likely to have suffered self-interference. One challenge here is that as the window size increases, the packet train becomes larger reducing gaps between the last packet of a train and the leading packet of the next train. As soon as the channel becomes full with packets, trains in successive cycles become contiguous with no time gaps. The identification of the leading packet pair (LPP) is done by estimating the expected number of packets in the congestion window on each RTT cycle (e.g. the fifth cycle, there will be 2^5= 32 packets being sent by the sender). Figure 2 (b) shows a small example where the gaps shrink and finally vanish after 4 cycles. From our experiments and for the range of file transfers that we encountered, the Leading Packet Pair method, in contrast to SRR, was able to identify in most of the experiments only 5 good packet pair samples. The advantage of LPP, compared to SRR is that no time regulation is needed to obtain good samples. However, the capacity estimation is computed based on the results of these 5 samples compared to 20 samples of the SRR. We explore the trade-off(s) between these algorithms in more details in the results section below.

B. Kernel instrumentation

For this work, we instrumented the Linux 2.6.18 TCP kernel module. The most important features were inserted directly into the "tcp_input.c" and "tcp_output.c" files concerning the receiver and sender machine state sides, respectively. We also modified the "tcp.h" file inserting new data structures to collect packet pairs into a constrained vector. The capacity calculation was implemented using a simple 64-bit division.

In dispersion measurements it is important to obtain precise timestamps of the arrival instants. Since, today's processors run at frequencies of more than 1GHz, we can use the instruction counter (RDTSC - Read Time Stamp Counter) to time events with higher accuracy and low overhead. In fact, the RDTSC CPU instruction allowed us to capture measures on the order of microseconds, providing high enough precision.

III. TESTBED DESCRIPTION AND EXPERIMENTS

In this section, we describe an extensive "path capacity estimation" campaign using our receiver implementations. Most of the experiments were conducted from 11/07/06 to 12/04/06 especially during the night time. The central measurement point was chosen to be UCLA (University of California Los Angeles). A number of measurements were collected exploring path diversity to a set of 34 open source web servers. This list of open source web servers was obtained from the site "Google Summer of Code 2006 [5]", a well-known repository of open source projects. From the initial list, we identified large downloadable files as preparation for the measurement study.

We installed our patched Linux 2.6.18 kernel in a high-end host, with a Dual Intel Xeon 3.2GHz processor, 1GB RAM, PCI-X 64B/133Mhz, Intel 1000 Server Pro NIC (1 Gbps Ethernet). The machine was connected directly on a Cisco Catalyst 4500, one of the main access switches to the UCLA all 10Gbps core backbone. The connectivity from the UCLA backbone to academic networks passes through CENIC (Corporation for Education Network Initiatives in California) with 1Gbps bottleneck, and 1Gbps to Abilene (Internet2). In addition to the careful choice of time and high capacity path, we ensured minimal limitation from the host system itself by deploying the Linux kernel in single user mode and stored all the logs and downloaded files in a RAMdisk of 64MB, reducing the likelihood of context switches and I/O requests.

An initial extensive validation of the path capacity estimation was done in several internal experiments within UCLA, assessing with remarkable precision several servers and clients at different IP addresses with capacities ranging from 2.5Mbps wireless LAN to 10Mbps and 100Mbps. In addition, experiments were conducted using UFES University in Brazil to connect to some local DSL modems (around 800 kbps).

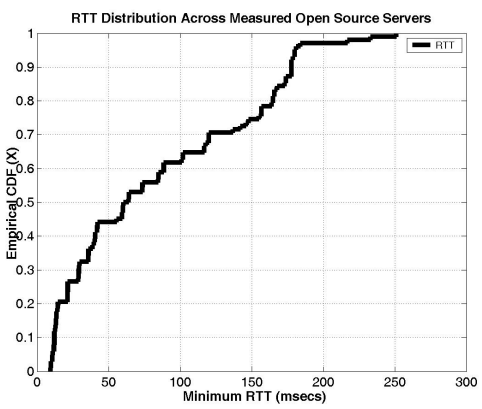
The bulk of open source server experiments consisted of more than 20,000 downloads from the chosen sites. Each download was limited to only 3 seconds, since we were interested in estimating path capacity during the first 40 packets transfer.

Our analysis of the collected data starts by presenting the heterogeneity of the chosen servers, in terms of propagation delay and path capacity. In figure 3(a), we observe that 65% of the websites had an end-to-end delay less than 100 msec, and therefore are most likely located in North America, or close to it. The remaining 35% had more than 100 msecdelay, capturing servers on other continents. In terms of narrow link capacity, Figure 3(b) shows that 75% of the measured paths to open source sites had narrow link capacities less than or equal to 100Mbps. The percentage of narrow links below or equal to 10 Mbps was about 30% of the cases. Other capacity groups obtained were in the 155 Mbps or OC-3 region (5%), in the 622.08 Mbps orOC-12 with (15%), and finally 5% of the paths were close to 800Mbps.

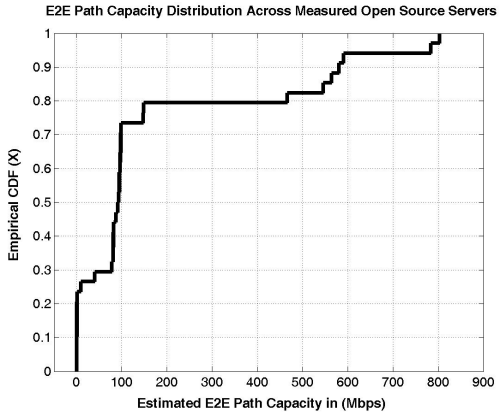
As we narrow down the estimates per server, using the short regulated method and 100 experiments per server, we group the estimations in small clusters (Figure 4(a)(b)(c)) with estimated links: below 100Mbps, from 100Mbps to 400Mbps and beyond 400Mbps. In each figure, we generate a boxplot presentation of the data for each group. The boxplot shows in a box area the main characteristics of the data distribution: the line in the center is the median of the data (less skewed centroid than average), the upper and bottom box limits represent the 25% and 75% quartiles.Finally, the points outside the box represent outliers (more than 3 times the standard deviation). The analysis of the results shows that the method has low variability in every group. Several of the capacities estimated here where confirmed by network administrators and ISP schematics.

The results "below 100Mbps" showed the smallest level of variation within each boxplot. The estimation of the "100Mbps to 400Mbps" narrow links presented some expected fluctuation due to cross-traffic, and possibly service policies [8]. Although low variability does not indicate high accuracy, we believe that it provides good evidence for the methods effectiveness. As we approach higher capacities, the measurements had a higher variability, as expected. This happens because any micro-level effect on the packet pair during its traversal through the end-to-end path can disperse the pair substantially. As a short example, if we are measuring a 1Gbps narrow link, the minimal packet pair dispersion is theoretically in the order of 10 μsec granularity, assuming 10,000 bit packets. While, 800Mbps theoretically should have as the minimal dispersion about 12.5 μsec, a difference of a mere 2 μsecs! We suspect that the higher variability due to high speed happens in the region above 622Mbps (OC-12), as it can be observed in Figure 4(c).

We now compare the inference methods: SRR and LPP. In order to compare them, we performed 21,332 experiments, to all sites, using each method. After the collection process, we computed the difference of the median obtained from each algorithm and every site. The results showed that the methods are nearly identical in their estimates, since the differences were smaller than 10Mbps in 81.25% of the cases.

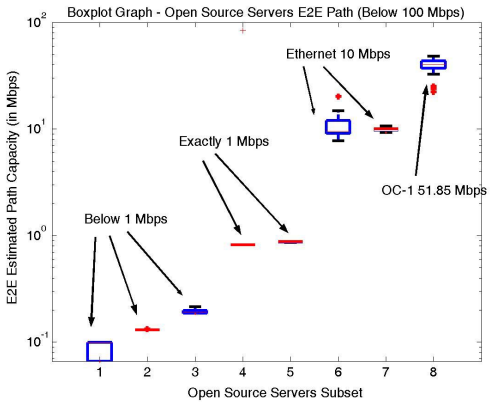


(a) RTT Diversity

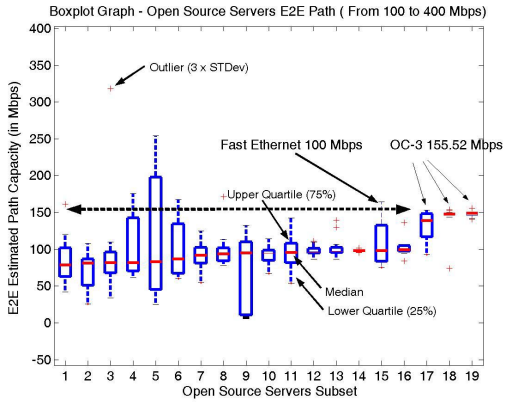


(b) E2E Path Capacity Diversity

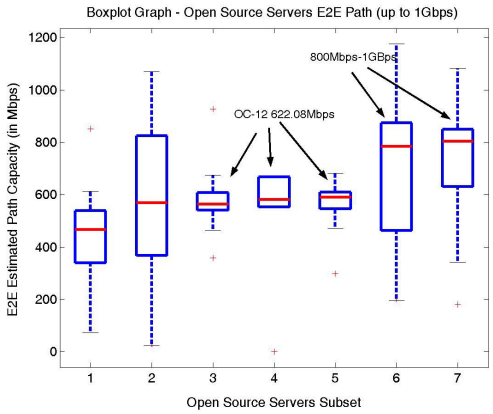
Fig. 3. Exploring the Heterogeneity of the Open Source WebServers



(a) Below 100Mbps Sites Subset

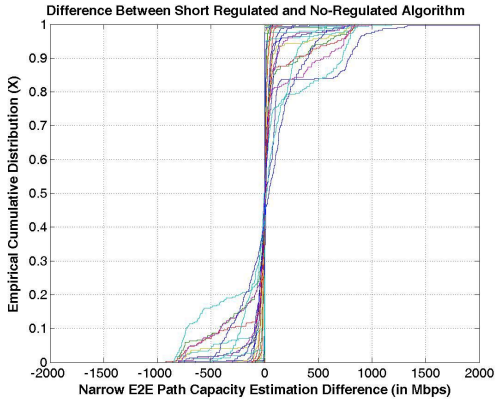


(b) From 100Mbps to 300Mbps

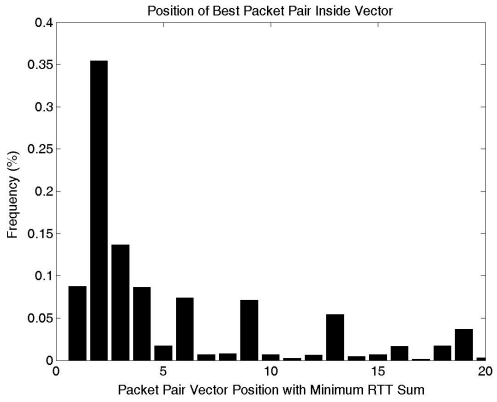


(c) Above 300Mbps

Fig. 4. Narrow Down the Link Capacities



(a) CDF Measurement Difference Per Site



(b) Best Packet Pair Position Distribution

Figure 5 – Narrow Down 2 Methods Difference

The remaining 18.75% can be viewed as outliers as can be seen from the distribution difference for each site in figure 5(a). The difference is calculated by subtracting every "SRR" measurement from its LPP equivalent. We observe that the method are not very different since for most of the cases the distribution difference was around zero. The only cases with non-zero difference are the ones where at high speed since some variability due to precision is expected, therefore not an artifact of the method differences. A thorough analysis of the position where the best packet pair occurred inside the vector that stores the packet pair estimates, showed a bias towards the first 5 packet pair samples. To repeat, "best packet pairs" are those with "minimal RTT sum" among all packet pairs of a flow. Thus the small difference observed between the algorithms is due to the fact that for 68.30% of the cases, the best packet pair was located within the first 5 packet pairs. This is illustrated in Figure5, where we present a histogram of the position where the best packet pair was found in the vector for our 20,000 experiments. The histogram shows a strong mode in the second packet pair of the connection, while the rest of the positions appear to be uniformly distributed. We argue that the bias in the first 5 packet pairs could be due to the expected low cross-traffic in the nigh time experiments, thus the first pairs would experience minimum RTT delay. In addition, the position of the strongest mode (Fig.5(b)) in the second packet pair, instead of the intuitive first packet pair, lies in the way applications fetch dynamic pages. The first packet pair will be subject to further processing delay while the second will go through directly. Finally, we investigate the likelihood of the minimum sum convergence (min{R1+ R2} = min{R1} + min{R2}). An analysis of our dataset showed that using the 20 packet pair vector, 72.98% of the samples converged. Moreover, the convergence has a strong filtering effect on the outliers of the capacity estimation distribution. This can be seen when comparing: samples without convergence and samples with convergence. This result is based on the fact that some experiments didn't run long enough to reach the Capprobe RTT sum convergence. Figure 6 shows the percentage of "good" samples estimated with and without minimum sum convergence (good here mean within a range of 2 * standard deviation from the median). Thus, it can be observed that the RTT min sum convergence increase the chance of obtaining "good" samples from 93% to 99.1%, on other words, the likelihood of false positive is reduced from 7% to 1%.

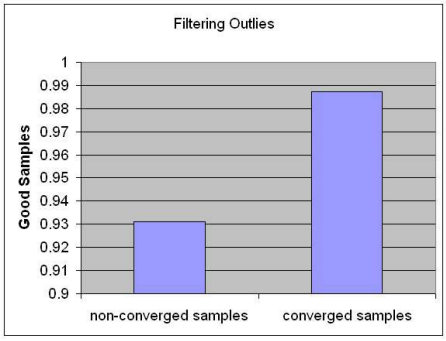


Figure 6 - RTT Sum Convergence Filtering Effect

IV. APPLICATIONS

In this section, we describe two applications of the receiver side capacity estimation methods we described above. In the first application, a TCP receiver tilizes the capacity estimate it obtains to better set the "advertised window size" it forwards to the sender. In this manner, the receiver effectively controls the amount of traffic in flight, and thus it reduces buffer occupancy at the bottleneck, and improves TCP throughput . The second application identifies the exact location of the narrow link (i.e. bottleneck) on an Internet path. It involves a combination of the *traceroute* and monarch [6] tools. We should emphasize that the purpose of this section is only to show the potential applicability of our embedded TCP capacity estimation. We did not perform an exhaustive experiments campaign, but we present some preliminary results.

A. Narrow Link Based TCP Flow Control

The developed application assures the proper use of the narrow link capacity, by controlling the TCP flow according to the maximum bandwidth-delay (BDP) product into a certain path. The BDP quantity is the maximum amount of packets or bytes than can be in-flight at any moment without using buffers; sometimes it is also called pipe size. In our proposed method the receiver sets the advertised window equal to the pipe size. Thus, reducing the usage of buffer space at the bottleneck. It is important to point out that, such flow control, TCP tries to fill out buffers in the bottleneck since it keeps increasing the sender congestion window until it hits a packet loss. An estimation of the optimal receive window, and a fast update of its value within the first 40 packets also impacts the slow start phase. A normal Linux implementation would start the receive window small (i.e. 4 packets), and increment by 2 for every packet. This potentially limits the effectiveness of the slow-start increment, since it prevents a large number of initial packets during the slow start. Our method allows the regular slow start to execute free of bounds until the pipe size is reached. We implemented the changes of receive update in the Linux kernel. Whenever the minimum RTT sum convergence is reached, we reset the receive window to an estimated value derived from the equation of Bandwidth delay product (capacity and minimum RTT) of the path. To evaluate the impact of this TCP receiver modification, in a controlled lab environment, we emulate typical DSL conditions using dummynet [9], a popular network emulation tool. We connected two machines through an emulated 2 Mbps link , set the buffer size to 300 packets, and set the propagation delay equal to 200 msecs. We configured a webserver as the TCP "sender" side and we used wget (a popular linux download application) to download a file from the server, as our TCP "receiver" side. We repeat the test using, on the same conditions: a normal Linux TCP and our capacity-based receive window modified version.

Figure 7 presents a solid line representing instantaneous throughput every 10 packets, while the dashed one is a cumulative average throughput from the beginning of the connection. The goal is to show that applying such limiting rate technique on an emulated environment improved the total throughput of the connection. Analyzing the modified case, we can observe that the instantaneous throughput is much regular/constant than the original standard one; in fact from 4 to 7 secs it is similar to a CBR. The cumulative throughput along the connection shows a large gap between the standard (170 KB/s or 1.36Mbps) and our modified version (225 KB/s or 1.8 Mbps), effectivelly improving the utilization of the 2Mbps link from 68% to 90%.

B. Narrow Link Location Tool

As we pointed out, this application uses a combination of traceroute and monarch [6] tool. The goal is to find the exact locations and capacity of a narrow link bottleneck on an Internet path. In addition, the tool was used to validate some local results at UFES network, estimating path capacities of non-cooperative wireless links, servers connected at 10Mbps hubs, servers at 100Mbps, and so forth accurately. The monarch tool was implemented to experiment with new TCP implementations in the "network at large" by forcing non-cooperative hosts (i.e. routers and "serverless" hosts) to reply to several types of probes as if it were a normal end-to-end TCP connection. To accomplish this task, Monarch starts a TCP connection with itself (with both call-legs, sender and receiver, in the same machine). However, the TCP packets, instead of being sent directly, they are changed into probes (ICMP, UDP, dummy TCP on closed ports) and sent directly to a specific remote host. The remote host then replies with errors/control messages (like ICMP replies or TCP RST). The final step changes the reply probes back to TCP packets as they arrive, this way, sending packets to the receiver side of the local TCP connection. In summary, we let *monarch* generate as much probing traffic as TCP normally would do (Figure 8)). Since our modified kernel is actually embedded in the TCP Receiver Side (*) as shown in Figure 8(a), it was not necessary to change the *monarch* implementation. We just instrument it by estimating path capacities from the local receiver to any non-cooperating host, in addition, exporting the estimate from the kernel to userland through a regular netlink() API. In order to perform the narrow link position discovery, our instrumented monarch tool calls traceroute to a certain destination, as shown in our execution example (Table I).

```
$ sudo ./caplimiter www.cs.caltech.edu monarch -p tcp-ack -z 1001
traceroute to whirlwind.cs.caltech.edu ( 131.215.44.115)
1 131.179.80.3 0.211 ms 0.191 ms 0.189 ms
2 131.179.12.3 0.708 ms 0.678 ms 0.672 ms
3 169.232.49.65 0.442 ms 0.403 ms 0.599 ms
4 169.232.4.22 0.580 ms 0.837 ms 0.429 ms
5 169.232.4.103 0.748 ms 0.738 ms 0.454 ms
6 137.164.27.5 1.089 ms 0.927 ms 0.900 ms
7 137.164.27.248 1.117 ms 1.067 ms 1.199 ms
8 131.215.254.43 1.368 ms 1.224 ms 1.244 ms
9 131.215.44.115 1.455 ms 1.368 ms 1.318 ms
Hop kudos-pb to 131.179.80.3 at 96412857 bps
Hop 131.179.80.3 to 131.179.12.3 at 98404081 bps
Hop 131.179.12.3 to 169.232.49.65 at 95985747 bps *
Hop 169.232.49.65 to 169.232.4.22 at 97530215 bps
Hop 169.232.4.22 to 169.232.4.103 at 98773510 bps
Hop 169.232.4.103 to 137.164.27.5 at 97373535 bps
Hop 137.164.27.5 to 137.164.27.248 at 97657516 bps
Hop 137.164.27.248 to 131.215.254.43 at 96416740 bps
Hop 131.215.254.43 to 131.215.44.115 at 81429147 bps *
Min capacity hop is 131.215.254.43 to 131.215.44.115 at 81429147 bps
```

Table 1 - Monarch Capacity Probe Execution Example

Once the intermediary routers are all known, the monarch tool is used to create one "emulated" TCP connection to each discovered router along the path. Thus, estimating step-by-step the narrow link capacity, and further its position, using a regular TCP loop.

V. CONCLUSION

In this paper, we presented a simple technique to estimate Internet paths capacity at a TCP receiver. Path capacity estimation has been widely explored in the literature. The novelty in this paper is embedding the estimator into a TCP receiver. We described our methods, tradeoffs of capacity estimate convergence whenever RTT minimum sum convergence is reached. In addition, we explored a rich set of open source software servers to discover the path capacities from a client to the set of servers.

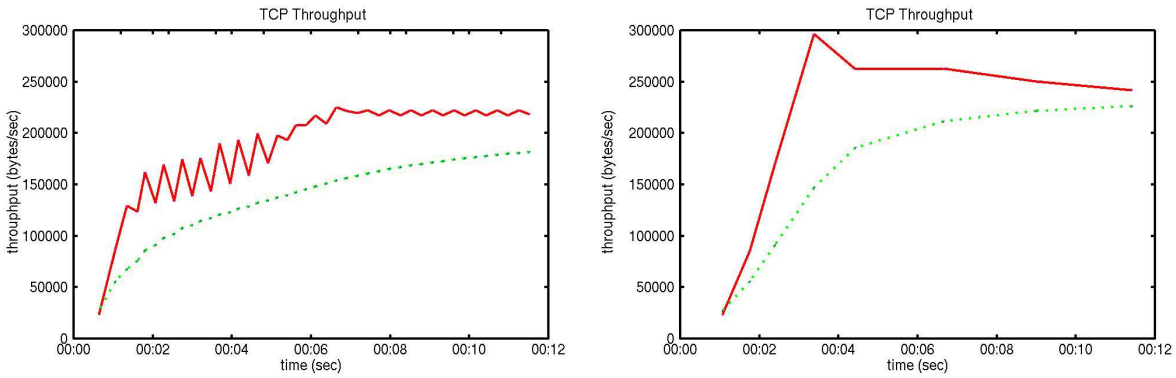
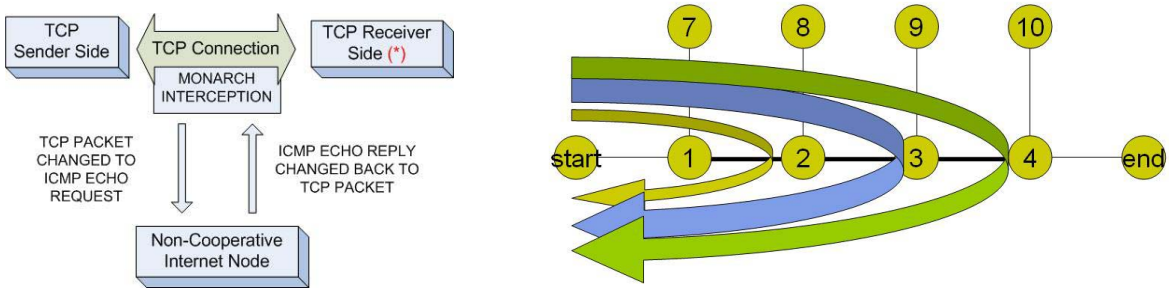


Fig. 7. TCP Throughput Comparison



(a) Monarch Design (b) Find the Position of a Narrow Link

Fig. 8. Discovering the position of Path Capacities on Any Host of the Internet

Validation was done by extensive tests in the lab, in the department local area network, and also by contacting some of the ISPs that we measured. The results present consistent accuracy up to 600 Mbps using the instruction CPU counter register. We presented two possible applications, one is an augumented version of Monarch [6] that narrows down the end-to-end capacity estimation to every router along a certain path. The other application is a TCP improvement where the receiver sets the advertised window equal to the estimate pipe size. The latter application was shown to improve TCP performance. Finally, we intend to further explore applications using the embedded capacity in the future.

ACKNOWLEDGMENT

The authors would like to thank Chris Frost (UCLA) for exporting path capacity to applications. In addition, this research was partially funded by grants NEC Systems Platforms Research Laboratories #57454 and FAPES (Fundacao de Apoio a Ciencia e Tecnologia do Espirito Santo) #31024866/2005.

REFERENCES

[1] R. L. Carter and M. E. Crovella. “Measuring bottleneck link speed in packet switched networks”. *Performance Evaluation*, 27:297–318, 1996.

[2] M. Crovella and A. Bestavros. “Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes”. *IEEE/ACM Transactions on Networking*, 6(5):835–846, 1997.

[3] C. Dovrolis, P. Ramanathan, and D. Moore. “Packet-dispersion techniques and a capacity-estimation methodology”. *IEEE/ACM Transaction on Networking*, 12(6):963–977, 2004.

[4] A. B. Downey. “Using pathchar to estimate internet link characteristics”. In *ACM SIGCOMM Computer Communication*, pages 241–250, 1999.

[5] Google. “Summer of code open source project”. In: http://code.google.com/soc/, 2006.

[6] A. Haeberlen, M. Dischinger, K. P. Gummadi, and S. Saroiu. “Monarch: A tool to emulate transport protocol flows over the internet at large”. In Proceedings of the *ACM/USENIX Internet Measurement Conference (IMC)*, 2006.

[7] R. Kapoor, L. Chen, L. Lao, M. Gerla, and M. Y. Sanadidi. “Capprobe: a simple and accurate capacity estimation technique”. In Proceedings of the *ACM SIGCOMM Computer Communication Review*, 34(4):67–78, 2004.

[8] V. E. Paxson. “Measurements and Analysis of End-to-End Internet Dynamics”. *PhD dissertation*, University of California, 1997.

[9] Luigi Rizzo. “Dummynet: a simple approach to the evaluation of network protocols”. In Proceedings of *ACM Computer Communication Review*, 27(1):31–41, 1997.

[10] Tcptrace. “Analysis of tcpdump files”. In: Tcptrace. http://iarok.cs.ohiou.edu/soflware/tcptrace/tcptrace.html/, 2001.

[11] B. Veal, K. Li, and D. Lowenthal. “New methods for passive estimation of tcp round trip times”. In Proceedings of *Passive and Active Measurements Conference (PAM)*, 2005.